

Contexto

Queremos evaluar tu capacidad para:

- Entender y abstraer un problema real.
- Diseñar una solución modular y mantenible en Python.
- Resolver problemas y manejar casos límite.
- Ser creativo/a al plantear mejoras o funcionalidades adicionales.

Para ello, te pedimos que desarrolles un **chatbot conversacional** que simule el proceso de compra en una tienda online utilizando **LangGraph**.

No buscamos un producto perfecto ni "de producción", sino ver **cómo piensas, cómo estructuras el código y cómo resuelves problemas**.

Alcance funcional

El chatbot debe permitir, mediante conversación en lenguaje natural, gestionar un **carrito de la compra** sencillo. Al menos debe soportar:

1. Catálogo de productos

- Define un catálogo pequeño (10–20 productos) en un JSON, lista en Python o similar.
- Cada producto debe tener: `id`, `nombre`, `precio`, y opcionalmente `categoría` o `descripción`.

2. Flujo básico de conversación

El usuario debe poder:

- Ver productos disponibles.
 - Ej.: "¿Qué productos tenéis?" → listar catálogo.
- Añadir productos al carrito indicando nombre o id, y cantidad.
 - Ej.: "Añade 2 camisetas azules" / "Quiero 1 producto 103".
- Eliminar productos del carrito o modificar cantidades.
 - Ej.: "Quita la camiseta" / "Pon 3 en lugar de 1".
- Consultar el estado del carrito.
 - Ej.: "¿Qué llevo en el carrito?" → listar productos y total.
- Confirmar la compra.
 - Ej.: "Quiero finalizar la compra".
- Solicitar y registrar datos mínimos de envío/facturación ficticios: nombre y ciudad, por ejemplo.
- Cerrar la conversación de forma clara.
 - Ej.: "Salir" / "Terminar".

3. Gestión de estados

- El chatbot debe tener en cuenta estados distintos, por ejemplo:
 - Navegación de catálogo.
 - Edición del carrito.
 - Confirmación de pedido.
 - Solicitud de datos de envío.
- Utiliza **LangGraph** para modelar estos estados/nodos y transiciones.

4. Manejo de errores/conductas inesperadas

- El bot debe reaccionar de forma razonable a:
 - Peticiones de productos que no existen.

- Cantidades inválidas.
 - Preguntas fuera de contexto ("¿qué tiempo hace?").
 - No hace falta resolverlo perfecto, pero sí mostrar **cómo lo planteas**.
-

Requisitos técnicos

- Lenguaje: **Python**.
- Framework conversacional: **LangGraph** (puedes usar LangChain si lo necesitas, pero el grafo de estados debe estar definido en LangGraph).
- Interfaz mínima:
 - Puede ser **línea de comandos (CLI)** o una pequeña API HTTP con FastAPI/Flask.
 - No es obligatorio un frontend gráfico, pero puedes hacerlo si quieres (bonus).

Organización del código:

- Estructura de proyecto clara (por ejemplo: app/ , models/ , graph/ , tests/ ...).
 - Separar lógica de dominio (carrito, productos) de la lógica conversacional (nodos, transiciones).
-

Entregables

Deberás entregar un repositorio (por ejemplo, en GitHub/GitLab) con:

1. **Código fuente** completo.
 2. **README.md** con:
 - Breve descripción del proyecto.
 - Instrucciones para instalar dependencias y ejecutar el chatbot.
 - Ejemplos de interacciones (copias de conversaciones o comandos).
 3. **Descripción de arquitectura** (puede ser en el README o en un doc separado):
 - Explicación de los nodos y estados del grafo de LangGraph.
 - Justificación de decisiones de diseño principales.
 4. **Tests básicos**:
 - Al menos 3–5 tests automatizados (por ejemplo, con `pytest`) que prueben:
 - Operaciones de carrito (añadir, eliminar, total).
 - Algún comportamiento de un nodo o transición clave.
 5. (Opcional pero valorado) Un pequeño **script o notebook de demo** o un archivo de registro de conversación.
-

Criterios de evaluación

Valoraremos:

1. **Comprensión y modelado del problema (20%)**
 - ¿Está bien representado el proceso de carrito y estados?
 - ¿Los flujos están claros?
2. **Diseño y arquitectura (25%)**
 - Organización del proyecto.
 - Separación de responsabilidades.
 - Uso correcto de LangGraph para modelar el flujo conversacional.

3. Calidad del código (20%)

- Claridad, legibilidad, nombres de variables, comentarios.
- Uso de buenas prácticas básicas en Python.

4. Robustez y manejo de errores (15%)

- ¿Qué pasa si el usuario hace cosas "raras"?
- ¿Hay estrategias de fallback / mensajes claros?

5. Creatividad y extras (10%)

- Niveles de detalle del catálogo.
- Funcionalidades adicionales (descuentos, validación extra, persistencia de carrito, etc.).
- Mejoras en la experiencia de usuario.

6. Documentación y pruebas (10%)

- README entendible.
 - Tests automatizados mínimos.
-

Plazo estimado

- Duración de la prueba: **2–3 semanas** (tiempo parcial, compatible con tus otras actividades).
- No esperamos dedicación full-time, pero sí un avance continuo y un resultado mínimamente pulido.